Self-adapting Security Monitoring in Eucalyptus Cloud Environment

Salman Mahmood[1], Nor Adnan Yahaya[2], Raza Hasan[3], Saqib Hussain[4], Mazhar Hussain Malik[5], Kamal Uddin Sarker[6]
School of Information Technology, Malaysia University of Science and Technology, Selangor, Malaysia[1, 2]
Computing and Information Technology, Global College of Engineering and Technology, Muscat, Oman[3, 4]
Department of Computer Science and Creative Technologies, University of West of the England Bristol, Bristol, England[5]
Department of Computer Science, American International University Bangladesh, Dhaka, Bangladesh[6]

***Abstract*—This paper discusses the importance of virtual machine (VM) scheduling strategies in cloud computing environments for handling the increasing number of tasks due to virtualization and cloud computing technology adoption. The paper evaluates legacy methods and specific VM scheduling algorithms for the Eucalyptus cloud environment and compare existing algorithms using QoS. The paper also presents a self-adapting security monitoring system for cloud infrastructure that takes into account the specific monitoring requirements of each tenant. The system uses Master Adaptation Drivers to convert tenant requirements into configuration settings and the Adaptation Manager to coordinate the adaptation process. The framework ensures security, cost efficiency, and responsiveness to dynamic events in the cloud environment. The paper also presents the need for improvement in the current security monitoring platform to support more types of monitoring devices and cover the consequences of multi-tenant setups. Future work includes incorporating log collectors and aggregators and addressing the needs of a super-tenant in the security monitoring architecture. The equitable sharing of monitoring resources between tenants and the provider should be established with an adjustable threshold mentioned in the SLA. The results of experiments show that Enhanced Round-Robin uses less energy compared to other methods, and the Fusion Method outperforms other techniques by reducing the number of Physical Machines turned on and increasing power efficiency.**

***Keywords*—Component; VM scheduling; cloud computing; Eucalyptus; virtualization; power efficiency; self-adapting security monitoring system; tenant-driven customization; dynamic events; adaptation manager; master adaptation drivers**

I. INTRODUCTION

Cloud computing is a technology that provides on-demand access to a pool of resources (such as networks, servers, storage, applications, and services) through networks. It is offered by companies like Google, Amazon, and SalesForce and eliminates the need for users to handle administration and IT maintenance [1]. Resource scheduling and allocation can be challenging for cloud providers due to the dynamic behavior of services and multiple types of cloud systems available [2]. Cloud computing is a form of computing as a service rather than a product, providing customers with access to software, resources, and information as a utility. It is cost-effective, with lower upfront costs and a pay-per-use model [3]. Virtualization technology is used by cloud providers to improve cost-efficiency and energy efficiency. Cloud computing is used in various applications such as website hosting, scientific methods, customer relationship management, and high-performance computing [4].

Server virtualization allows for the allocation of computer resources (such as CPU and RAM) on demand through a pay-as-you-go model, where clients (tenants) only pay for what they use. Infrastructure as a Service (IaaS) is a popular cloud model using virtual machines (VMs) and virtual networks to provide tenants access to compute, storage, and network resources. By outsourcing some information systems through the virtual infrastructure on the cloud provider's physical infrastructure, businesses can enjoy automated management, flexible resource allocation, and the illusion of unlimited computing and networking capabilities, as outlined in the Service Level Agreement signed by tenants and the cloud provider.

Despite the potential cost and efficiency benefits of cloud adoption, security remains a major concern. Multi-tenancy, an essential aspect of cloud architecture, enables the coexistence of trustworthy and hostile virtual machines, making the cloud vulnerable to attacks from both inside and outside the environment [5]. A successful attack could lead to alteration of data stored in the cloud, including login credentials, and even complete control of the cloud infrastructure for illicit purposes. Traditional security solutions like traffic filtering and inspection are insufficient against sophisticated threats targeting virtual infrastructures. To ensure cloud security, an automated self-contained security architecture incorporating multiple protection and monitoring technologies is necessary [6].

In Infrastructure as a Service (IaaS) cloud architecture, tenants are responsible for managing their virtual information systems while the provider manages the physical infrastructure. Tenants have concerns about security monitoring of their virtualized infrastructure and need a solution that considers their specific security requirements and can respond to dynamic events in the cloud environment. This research aims to create a self-adaptable security monitoring framework to address these concerns and ensure adequate security monitoring for tenants' virtual infrastructures. The research work focuses on creating a self-adaptive security monitoring framework for cloud infrastructure. The framework should respond to dynamic events in the cloud and adjust its components accordingly, while maintaining a balance between security, performance, and cost. It should incorporate tenant-

driven customization and meet tenant-defined thresholds and security requirements. The framework should not create new vulnerabilities and should not significantly impact performance or regular cloud operations. The research work also assesses existing cloud computing approaches and scheduling methodologies, and explores Eucalyptus cloud scheduling methods. Two independent Eucalyptus virtual machine scheduling techniques are proposed and evaluated, which aim to improve energy efficiency in cloud data centers.

Cloud computing is becoming increasingly prevalent in various industries, and efficient resource allocation and security are critical for maintaining acceptable throughput and revenue. Therefore, it is important to evaluate and compare existing VM scheduling algorithms and develop a comprehensive self-adapting security monitoring system that meets the specific needs of each tenant in a cloud environment. While there are many existing virtual machine scheduling algorithms for cloud computing environments, the specific context of the Eucalyptus cloud environment has not been extensively studied. Therefore, this paper aims to evaluate and compare existing VM scheduling algorithms in the Eucalyptus cloud environment, with a focus on energy efficiency. The paper addresses are the lack of a comprehensive self-adapting security monitoring system that takes into account the specific requirements of each tenant in a cloud environment. The paper proposes a framework for such a system that combines precise security monitoring with self-adaptation.

This paper is organized as follows: Section II presents a literature review and an overview of related works in this field. Section III presents the design and methodology used in the study. Section IV presents the discussion on the study and results. Finally, Section V draws a conclusion and proposes future research.

II. Literature Review

A. New Approach to Cloud Computing

Cloud computing is a software-based network infrastructure that enables users to access and store data on demand. It allows for flexible, elastic and cost-effective use of IT resources without the need for new hardware or software. The five core properties of cloud computing include independence, resource pooling, on-demand self-provisioning, rapid adaptation, and a consistent network as shown in Fig. 1. Cloud computing also includes three delivery options: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). There are four deployment strategies for cloud computing: public, private, communal, and hybrid clouds [7].

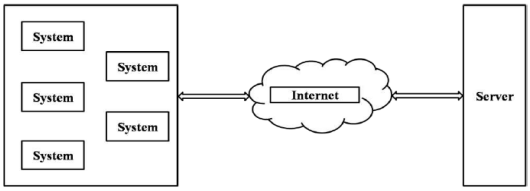

Fig. 1. Basic architecture of cloud.

B. Cloud Computing Models based on Services

The Cloud Security Alliance has identified security and privacy concerns as major obstacles to trusted cloud computing. Different security levels are needed for public and private clouds and Service-Level Agreements (SLAs) define customer and cloud provider responsibilities. Key safeguards include data integrity, vendor trust, consumer confidentiality, individual users and user groups. There are three delivery methods for cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS provides a layer of middleware and OS services, while PaaS offers a web-based platform to run apps as shown in Fig. 2. SaaS is the pinnacle of cloud computing technology, provided as a service and accessed online [8]. PaaS has the advantage of cost-effectiveness, but there is a risk of lock-in if it uses proprietary interfaces or programming languages.

C. Cloud Computing Security Issues and Challenges

Cloud computing is a rapidly growing field in IT that provides benefits such as improved infrastructure and cost-efficient access to services. However, it also presents risks such as security threats, privacy concerns, and reliability issues [9]. To address these, there is a move towards open standards, better compatibility, and compliance resources from cloud service providers. When considering cloud computing, it is important to also consider its long-term viability. There are challenges to cloud computing solutions such as lack of interoperability, compatibility with existing programs, difficulty in meeting regulations, and insufficient security. Lack of standardization and proprietary applications can lead to complexity and high costs, as well as security concerns with shared infrastructure not providing enough security [10].

D. Resource Allocation

Network resources or shared resources can include files, the challenge in sharing resources such as data, audio and video, and hardware in a cloud environment can be addressed through resource management. This involves coordinating IT resources by controlling templates, managing virtual IT resources, performing load balancing, resource replication and failover, and monitoring the operational conditions of IT resources [11]. The cloud provider or user accesses these functionalities through a cloud resource administrator, and the virtual machine image repository is located in the Virtual Infrastructure Manager (VIM) as shown in Fig. 3.

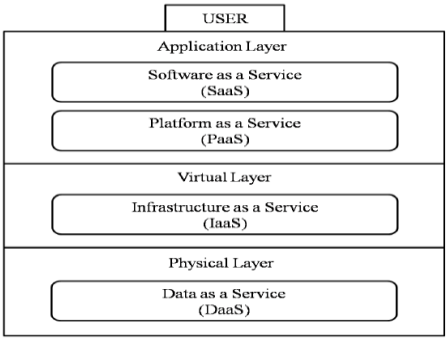

Fig. 2. Service model of cloud computing.

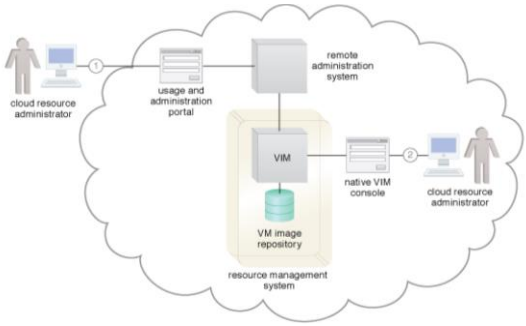

Fig. 3. Resource management architecture.

E. Primary IaaS Systems

The OpenStack platform is a popular open-source cloud management system that is used by tenants to access, build, and manage their resources through a web interface or command line clients [12]. The system is designed to be modular, with a central controller node and compute nodes that report on the status of deployed virtual machines as shown in Fig. 4. The cloud infrastructure is dynamic, and changes may occur at various levels including services, topologies, and traffic. There are four types of cloud deployment models - private, public, community, and hybrid clouds. In a cloud environment, tenants can easily deploy virtual machines and create services that are accessible to others [13]. The cloud infrastructure is dynamic and changes happen frequently, including topology-related events caused by virtual machine life cycle commands, and traffic-related events due to changes in network demand.

F. Virtualization

The three main architectural layers in an IaaS system are the physical layer, the hypervisor, and the virtual machine layer. The security monitoring architecture focuses on the virtual machine layer. There are four main server virtualization strategies: emulation, full virtualization, par virtualization, and OS-level virtualization. OS-level virtualization uses containers and is a popular option for minimal overhead. Network virtualization is crucial for IaaS cloud architecture and manages IP addresses and communication. Network virtualization is achieved through MPLS, VLANs, Flat Networking, and GRE encapsulation, and is simplified by Software Defined Networking (SDN), with OpenFlow being the most popular example. The SDN architecture separates the control and data planes and allows for centralized control of the network [14][15] as shown in Fig. 5.

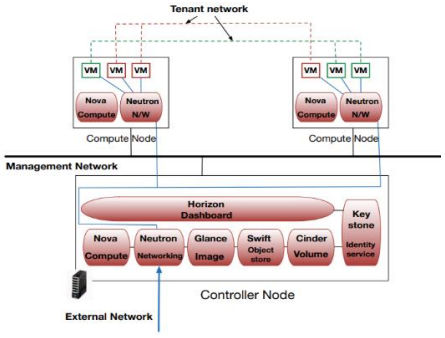

Fig. 4. The modular architecture of openstack.

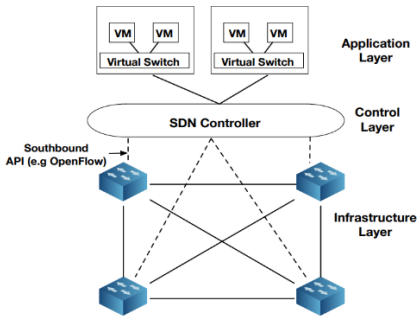

Fig. 5. The architecture of SDN.

SDN (Software Defined Networking) is a network management concept that allows for programmatic control and administration of the network through a programmatic interface. Examples of SDN controllers include OpenDaylight and Floodlight. Network virtualization in IaaS (Infrastructure as a Service) clouds allows for programmatic creation, modification, and deletion of network objects (such as networks, subnets, ports, etc.) without affecting the underlying hardware infrastructure [16]. The Neutron component of OpenStack is responsible for managing tenant networks and providing VMs with networking capabilities, with the ML2 plugin creating virtual bridges to connect VMs to networks with either GRE encapsulation or VLAN tagging to differentiate network traffic among tenants. In a typical cloud implementation, three types of networks are created: management network, tenant networks, and external network.

G. Security Threats

Different types of security threats to information systems, including the application level, network level, operating system level, and cloud environment. Threats can include SQL injection attacks, cross-site scripting, buffer overflows, impersonation, denial of service attacks, man-in-the-middle attacks, and exploitation tactics. In cloud environments, both tenants and providers face security risks such as risks from other tenants, the provider's infrastructure, and the API. Multi-tenancy can also create new security threats, such as side channel attacks and exploitation of shared resources. The security of virtual machines in a cloud environment is dependent on the stability of the hypervisor, which can be a target for security vulnerabilities and malware attacks. A successful attack on the control interface can result in full compromise of the account and stored information [17].

H. Security Monitoring

Information systems are continuously threatened at several layers of their infrastructure. To prevent significant damage, it's crucial to have a security monitoring system in place, such as an Intrusion Detection System (IDS). IDSs are the main component of a security monitoring framework and they detect security holes by collecting, processing and reporting data. There are two types of IDSs: signature-based and anomaly-based. Signature-based IDSs use string comparison to match observed events to known attack patterns, while anomaly-based IDSs compare observed events to a normal profile of activity to detect potential security breaches. Additionally, there are two types of IDSs based on location: network-based

and host-based. Network-based IDSs monitor network traffic, while host-based IDSs monitor a single host for suspicious activity [18]. In cloud environments, security monitoring must be automatic to adjust to changing events. Different cloud security monitoring strategies focus on either the tenant's data system or the provider's infrastructure. Provider infrastructure monitoring includes hypervisor or kernel-based IDSs as shown in Fig. 6. The trend in hypervisor security is to lower the Trusted Code Base, but this does not guarantee the complete integrity of the system.

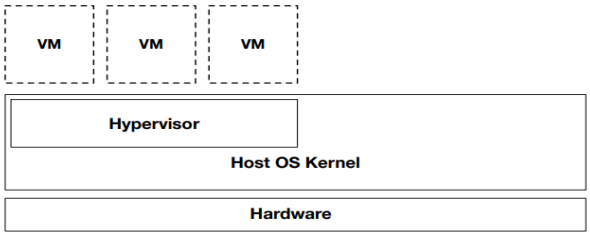

Fig. 6. The kernels of the host and hypervisor.

In the context of security monitoring in cloud environments, there are two main areas of focus: the provider's infrastructure and the tenant's information system. The provider's infrastructure is monitored by hypervisor or kernel-based intrusion detection systems (IDSs), which focus on securing the integrity of the hypervisor and kernel [19]. However, these systems may not adapt to changes in the programs running in the monitored system. On the other hand, the tenant's information system can be monitored through virtual machine introspection, which allows for real-time monitoring of the health of the underlying operating system and active processes in deployed virtual machines. However, designing cloud-tailored intrusion detection systems for the tenant's information system can be challenging due to the complexity and variety of the cloud environment and the conflicting security requirements of tenants.

It is evident from the literature review that there are still several areas that require further research, including the development of standardized SLAs, the development of cloud computing architectures for a wider range of applications, the effectiveness of cloud computing security measures, and the environmental impact of cloud computing.

III. Design and Methodologies

Eucalyptus is an open-source software platform used to create an IaaS (Infrastructure as a Service) for private or hybrid cloud settings. It is scalable and distributed, with each component serving a small number of users, making it suitable for enterprises of all sizes. The platform offers virtualized cloud resources like infrastructure, network, and storage as a service. The name "Eucalyptus" stands for Elastic Utility Computing Architecture for Linking Your Programs to Useful Systems.

A. Setting up Eucalyptus Cloud

Eucalyptus cloud setup consists of the Cloud Controller (CLC) and Walrus, which manage various clusters of real computers that host virtual instances. Each cluster has a Cluster Controller (CC), a Storage Controller (SC), and multiple physical machines known as Nodes [20]. The Node Controller regulates the hypervisor on each node to manage virtual instances. In the study mentioned, all components were co-located on the same system except the NC, with one machine hosting CLC, Walrus, CC, and SC, and five machines hosting NC as shown in Fig. 7.

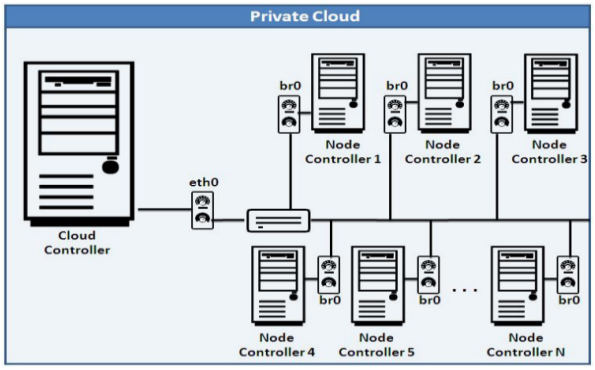

Fig. 7. The NC service is hosted by the domain-0 kernel in the xen setup.

B. IaaS Clouds based Self-Adaptable Framework for Monitoring Security

We consider an IaaS cloud environment with a global cloud Tenants have control over a networked group of virtual machines (VMs) and can specify unique monitoring requirements through a Service Level Agreement (SLA) or API. The cloud controller provides networking capabilities and the tenant receives both an external and internal IP address. The study focuses on software attacks from inside or outside the cloud infrastructure and the potential for an attacker to exploit a deployed VM and compromise the victim's infrastructure [21]. Trust is placed in the cloud provider's infrastructure being physically safe and not affected by malicious viruses. Attacks that weaken the cloud management system are not taken into account.

C. Goals Designed

The objective of the research is to develop a self-adaptive security monitoring platform for virtualized information systems of tenants that meets several requirements including self-adaptation, tenant-driven customization, security and accuracy, and cost savings. The framework should be able to update its components automatically in response to changes in the cloud environment and to allow for customization based on tenant needs [22]. It should minimize security risks and costs for both the provider and the tenants. The framework should consider the different sources of adaptation, including changes in services, topology, monitoring load, and tenant requirements. The framework should ensure that reconfigurations do not compromise security or monitoring accuracy, while minimizing resource use and performance impact on tenant applications.

D. Methods

The research presents a self-adaptive security monitoring framework for IaaS cloud with a three-tier architecture including a controller, two computing nodes, network IDS, edge firewall, local firewalls, and a log aggregator. The three

tiers are the tenant, adaptability, and monitoring devices with the monitoring devices consisting of log collectors, aggregators, and probes as shown in Fig. 8.

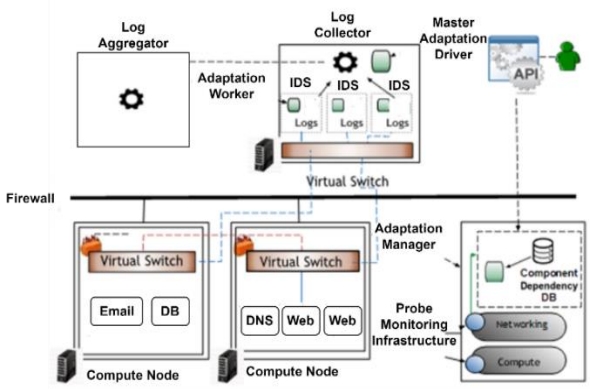

Fig. 8. The framework's architecture [23].

The research presents a self-adaptive security monitoring framework for IaaS cloud with three main tiers: tenant, adaptability, and monitoring devices. The adaptation level is responsible for planning and enforcing the adaptation process and is made up of the Adaptation Manager in the cloud controller, Master Adaptation Drivers in the nodes, and a dependency database. The Adaptation Worker in each monitoring device enforces the reconfiguration settings and the Infrastructure Monitoring Probes discover topology changes. The tenant API provides access to all monitoring features as shown in Fig. 9.

The research presents a self-adaptive security monitoring framework for IaaS cloud with a controller, two computing nodes, network IDS, edge firewall, local firewalls, and log aggregator. The framework has three main tiers: tenant, adaptability, and monitoring devices. The tenant tier has access to a tenant API, which allows tenants to express their monitoring requirements in a high-level language. The API is broken down into two parts: the tenant-exposed part and the translation part. The tenant-exposed part allows tenants to access the list of monitoring services, add or remove a monitoring service, and alter monitoring metrics. The translation part translates the high-level tenant requirements into framework-specific information, which is used by the framework to make adaption decisions.

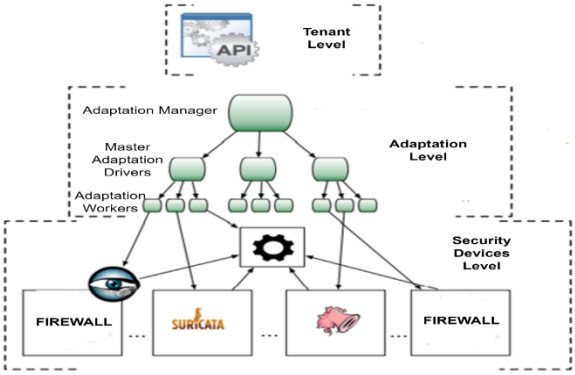

Fig. 9. The framework's various levels [23].

Example scenario: Consider the following instance as an illustration: On various compute nodes, there are two unique VMs installed, each of which belongs to a different tenant. An ssh server and an Apache server with SQL support are hosted by the first VM with ID 24 that is installed on node A. Although its private IP address is 192.168.1.3, the virtual machine's public IP address is 182.12.34.201. A port with the name "qvo1432" is used to link the VM to the compute node's virtual switch. On node P-20, the second VM, ID 63, is configured and is only used to host an ssh server as a service. The VM's secret IP address is 192.168.1.3, whereas its public IP address is 182.12.34.199. In this condensed example, we solely use firewalls and network-based IDSs as monitoring tools.

The security of a cloud computing infrastructure is monitored by security devices such as firewalls, vulnerability scanners, antivirus programs, and others. These devices are able to create log files that are managed by log collectors and aggregated by the framework administrator for searching for specific patterns. The Adaptation Manager is a key component of the monitoring framework that is responsible for selecting the adaptations to the monitoring tools, maintaining an acceptable degree of monitoring, and managing dynamic events within the cloud architecture. The Adaptation Manager performs algorithms in Algorithm 1 in response to dynamic events and makes decisions on whether to adapt based on the topological and functional overviews of the monitoring framework.

Algorithm 1 The choice algorithm for adaptation

```
1: procedure ADAPTATIONS (dynamic activity)
2:     services list ← MAP (dynamic activity.VMid, vm info file)
3:     affected equipments, agents ← MAP (dynamic activity.VMid)
4:     for j in affected equipments do
5:         reconfiguration needed ← DECIDE (j, services list)
6:     PROPAGATE DECISION (agents, reconfiguration needed)
```

- Connect the ID of the VM that is affected by the modification to the list of services that are currently executing within the VM (line 2 in Algorithm 1). To do this, the data in the file that the API generated is parsed.
- Determine the monitoring systems in charge of the impacted VM. These will be the monitoring tools that are modified. Utilizing data from the Component Dependency Database, this is accomplished. The vm information file is a single file that contains both the list of monitoring devices that will be updated and the list of active services.
- Select the necessary reconfiguration type (line 5 in Algorithm 1). Different reconfiguration kinds may be required depending on the monitoring device type and incident category.
- Distribute the reconfiguration parameters to the personnel in charge of upholding the adaption choice (line 6 in Algorithm 1).

E. Infrastructure Monitoring Probes

IMPs (Instance Monitoring Processes) are components of the cloud engine that monitor topology changes, such as placement and VM lifecycle alterations. They collect VM-related data from the cloud engine and provide information to the adaptation manager, which then decides which adaptation to deploy. IMPs do not affect normal cloud operations when adjusted.

F. Component Dependency Database

The article discusses the challenges of security issues in complex security monitoring frameworks composed of various components. The methodology states that a decision taken in response to a dynamic event can impact both an active and passive monitoring device, requiring both to be changed. The Dependency Database, a component of the cloud controller, lists all security devices for each monitored VM and gives the Application Manager both functional and topological viewpoints. The VM information table can be used by the Adaptation Manager as a key to access the list of monitoring tools in charge of a VM, making it easier to identify impacted devices during an adaptation. The VM information table for the VMs in the previously discussed scenario is found in Table I.

For example, the VM with ID 24 can see a host IDS, a network IDS, and two firewalls: one inside the local switch called f-p-20 and one on the edge called f-ext1. Multiple IDS types can monitor a single VM (host- and network-based).

TABLE I. THE VM INFO TABLE

VM ID	Network IDS	Host IDS	External-firewall	Switch-firewall
24	S-79	24	Mar24	f-p-20
14	S-99	14	Aprs14	f-p-63

TABLE II. THE EQUIPMENT INFO TABLE

Equipment Name	Location	Equipment Type
S-65	182.12.34.201	signature based

Device-specific information is kept in the equipment info table. The Adaptation Manager gathers the following data using each device name: both the device's location and its type Table II contains the S-65 IDS equipment information table. This instance demonstrates that the signature-based NIDS suricata65 is situated on a node with the IP address 182.12.34.201. The Dependency Database's information is used by the AM to compile a complete list of all devices that are impacted by a dynamic event. The AM updates the two tables with a corresponding entry for every new monitoring device that is created, including all the relevant data.

G. Self-Adaptable System for Intrusion Detection in IaaS Cloud

A self-adaptable intrusion detection system has been proposed for IaaS cloud environments. This system offers features such as self-adaptation, customization, scalability and security, and correctness. The system allows for adaptation to changing conditions in the cloud, enables tenants to modify monitored events, and adjusts the number of IDS systems based on network traffic and infrastructure size. The system ensures proper level of detection during adaptation to maintain security and correctness.

1) Proposed framework: Detailed design of a proposed framework, including its components and the system and threat model used. It starts with a general overview of the framework, followed by a detailed explanation of how each component works.

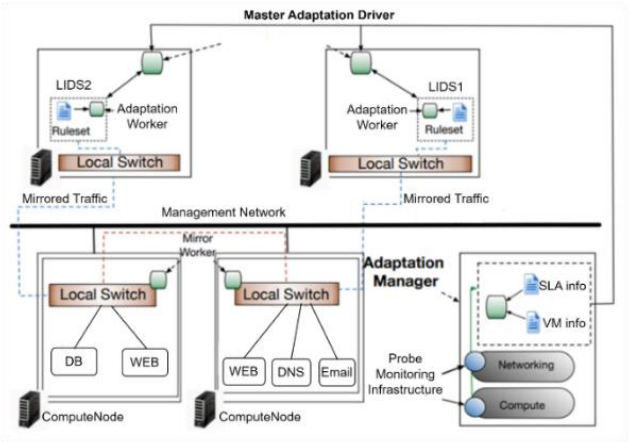

Fig. 10. The proposed framework for self-adaptable system.

The framework described in Fig. 10 is composed of four main parts: Adaptation Worker (AW), Local Intrusion Detection Sensors (LID), Mirror Worker (MW) and Master Adaptation Driver (MAD). LIDs are deployed on separate nodes and are used to collect and analyze network packets, either using anomaly-based or signature-based techniques. The AW updates the applicable ruleset, monitors the performance of the LID, and updates the MAD after a successful reconfiguration [24]. The MAD handles the reconfiguration of multiple LIDs, manages their lifecycle, and collects performance metrics. The MW ensures accurate mirroring of traffic to the matching LID node and constructs a mirroring endpoint if needed. A safety mechanism on each compute node ensures that VMs don't become active before the accompanying LID has been reconfigured.

The proposed framework for security in cloud computing has both potential weaknesses in its architecture and contributions to the service provider's infrastructure. A vulnerability in the framework is the configuration files that translate adaptation arguments into rule names, which are simple text or XML files and could be manipulated by attackers to produce fake adaptation arguments. The framework includes parsers for these files. The power utilization model evaluates the power usage of physical machines and is based on power meters built into the physical machines [25]. Research has shown that power consumption is inversely correlated with the number of fully utilized cores. The migration model describes the methods and costs of relocating virtual machines as shown in Fig. 11.

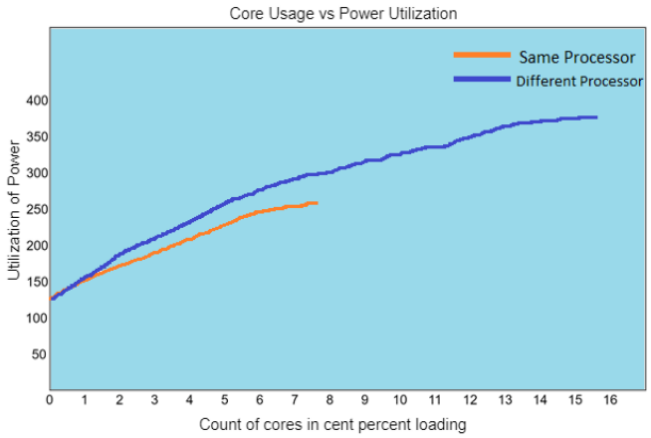

Fig. 11. Power utilization demonstration with various processor loads.

The Top power (Tp) is the maximum power used by a PM (Processing Module) when all 16 cores are fully loaded. The Inactive power (Ip) is the power consumption when none of the cores are loaded. The Top power is about half of the Inactive power. The linear power model is used to fairly assess the power utilization (PU) of PM.

$$PU = \left[\frac{C_v}{C_p}(1-\beta)+\beta\right]T_p \quad (1)$$

Where β displays the ratio of dormant to active power. Consider Fig. 11 above, β is set to half, Cv is the total number of cores required by resident VMs, and Cp is the total number of cores required by PM.

The Migratory technique uses live migration to allow a server administrator to move an active virtual machine (VM) to a different PM without affecting its performance. Although live migration does not affect the execution time of the VM, it does increase energy consumption during the transfer due to increased load on the receiving PM [26]. The energy consumption during relocation has been quantified based on experimental data.

$$U_E = U_S + U_D \quad (2)$$

$$U_S = \left[\frac{C_v}{C_p}(1-\beta)+\beta\right]T_p M_t \quad (3)$$

$$U_D = \left\{\left[\frac{C_v'}{C_p}+L\right](1-\beta)+\beta\right\}T_p M_t \quad (4)$$

The U_E is the percentage load of the sending and receiving machines. Cv is the number of cores required to migrate a VM, and Cv/Cp and Cv'/Cp are the percentage load of the sending and receiving machines. The migration time (Mt) and additional burden (L) imposed by the migration procedure must be considered when deploying VMs with various hardware requirements on PMs with specific hardware limitations. The First-Fit approach may be used to address the fusing problem, though it is more challenging. It seeks to deploy a VM to the first PM that has space for it and can be simple to apply, but may not be the best solution.

This study investigates a more challenging VM fusing problem where each VM has resource requirements, Time of Execution and Time of Arrival restrictions, and a limited availability window during its execution. The idle cores used by a VM after it has ceased to function can lead to the need for additional PMs. The Eucalyptus architecture provides two scheduling options, Enhanced Round-Robin (ERR) and a Fusion mechanism, to lower the number of PMs needed and save power consumption. The ERR approach uses two criteria to aid in VM fusion: retiring PMs cannot have additional VMs added to them, and retiring PMs that cannot complete all VMs before the deadline must transfer and shut down. The Fusion method combines the ERR and First-Fit approaches, depending on the rate of incoming VMs, to save more energy. The biggest challenge in implementation is choosing a reasonable Limit of Resigning (LR), calculated by adding migration time and time left for the VM to execute. The Fusion method with LR calculation can result in energy savings by transferring VMs during non-peak hours.

$$U_S = \left[\frac{C_v}{C_p}(1-\beta)+\beta\right]T_p R_t \quad (5)$$

$$U_D = \left[\frac{C_v'}{C_p}(1-\beta)+\beta\right]+T_p R_t \quad (6)$$

Where U_E and U_D are, respectively, the energy expenses associated with sending and receiving PMs when the VM is not being migrated. Instead of deciding to relocate every VM by its remaining time for execution after the PM resigned, Rt, that's a mystery, the limit α below is the resignation limit.

$$\alpha = (1-L)M_t + \frac{L}{\beta} \quad (7)$$

Assuming a VM with Rt smaller than indicates that not migrating saves more energy, and the VM will eventually finish before PM starts moving VMs. On the other hand, in the unlikely event that Rt greater than α, VMs will be transferred after the resigning limit is exceeded, indicating that moving is the better option. Execution time that is left over Rt While using the ERR approach, it is not essential. The limit α has been established and fixed. After a brief pause, α., a resigning VM will be shut down. Regardless of how much time is left for execution, each incomplete VM would be transferred.

2) Self-Adaptable framework for monitoring security: The proposed framework consists of four main components: Adaptation Worker, Local Intrusion Detection Sensors, Mirror Worker, and Master Adaptation Driver. The Adaptation Manager, a part of the proposed framework, is implemented using a multi-threaded model in Python, using Open-Stack as the cloud management system [27] and Open vSwitch for network traffic mirroring. The AM receives notifications of topology changes from the Infrastructure Monitoring Probes and creates a worker thread to manage the potential adaptation event as shown in Listing 1. The worker thread reads information about the impacted guest VM from a vm information file and retrieves the list of active services and tenant-specific security requirements using the VM ID as an identifier.

Listing 1 Adaptation during VM migration

```
1: procedure ADAPTATION (VM network information)
2:     SPAWN ADAPTATION THREAD
3:     services list ← INFORMATION PARSER (VM network
info.VM id, vm information file)
4:     affected equipments, locations ← INFORMATION
PARSER (VM network info.VMid, VM network info.source
node,VM network info.destination node, topology.txt)
5:     for p,qj in affected devices, locations do
6:     args.txt ← DECIDE (services list, p)
7:     IDS CONN (q, args.txt, +/-)
```

The parameters (such as what types of rules will be activated/deactivated, what is the acceptable tenant drop rate, etc.) are put to a specific file called args.txt when the AM decides to adapt. A separate file (topology.txt) that contains the topological and functional views required by the AM is extracted by the worker to obtain the names, kinds, and locations of the impacted security probes.

One NIDS is used on each computing node in the monitoring technique shown in this section. The single node hosts all of the deployed NIDSs. Following a VM migration, the master thread receives network-related information from the IMP, for example for the VM with ID 24.

The worker thread parses the topology.txt and vm information file.xml files as soon as it receives this information to extract the services that are currently running in the migrated VM (sshd, apache2, sqld), any additional tenant-defined monitoring requirements (worm), tenant-specific monitoring metrics, and finally the names of the NIDS responsible for monitoring the traffic in the source and destination nodes (S-9 and S-65, respectively), as well as their host IP. Adaptation is necessary for these two NIDS. The adaptation parameters are then written by the worker thread to adaptation args.txt. Listing 2 displays the findings of the NIDS monitoring traffic to and from the destination. Listing 2 shows the file holding an NIDS's adaption arguments.

Listing 2

```
1 s i g n a tu re  b a s e d
2 S- 6 5
3 apache 2
4 s q l
5 ssh 1 9 2 . 1 6 8 . 1 . 1 ,  1 9 2 . 1 6 8 . 1 . 3
6 worm
7 5
```

- Using a secure connection, the worker thread sends the dedicated file to a MAD in the node(s) holding the affected security devices. The name and IP address of the node housing the security device are used to create a connection by the ids conn function.
- The file containing the adaptation arguments must additionally contain a specialised operator if the adaptation calls for the activation or deactivation of monitoring parameters (such as + or -), as stated by the AM. In our illustration, a + denotes that the monitoring parameters need to be activated by the operator that was sent with the file in Listing 2. To allow for the concurrent transmission of the adaptation file, a separate thread is created for each security component that will be touched by the adaptation choice.

H. Self-Adaptable System for Intrusion Detection

A private cloud framework prototype was created using OpenStack and Open vSwitch as a virtual switch. The framework uses GRE tunnels to separate VMs into tenant networks and mirrors traffic using signature-based LID nodes in Docker containers [28]. An Adaptation Worker was developed to manage the LID nodes and communicates with the Master Adaptation Driver (MAD) using a shared folder. MAD uses a multithreaded method to manage and reconfigure the LID nodes and includes configuration files to convert adaptation parameters into rules for the IDS. The plug vifs function was used to delay the creation of virtual interfaces until the LID reconfiguration is finished to ensure network access for the VMs. [15].

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

The experiment was performed on an eight-node cluster, each with a quad-core CPU and a power model that uses half of peak power in idle mode. The cost of migration was calculated as 0.025 * 1/8 * 0.2. Data for VM arrival and execution times, and whether they were small or large-scale, was used in the test. The arrival and execution times were estimated using a normal distribution of two to ten hours. Small-scale tests were conducted to study power and migration models using the Xen hypervisor on each node. The power usage was measured using the proposed power model and actual power consumption and an average power usage was calculated. The results of the proposed ERR strategy were compared to the First-Fit power-saving strategy and showed little variance between the estimated and measured power. Thus, different scheduling techniques can be evaluated using the results of the recommended model as shown in Table III.

A simulation was conducted to evaluate the performance and energy consumption of a system with 500 octa-core servers and 3000 virtual machines. The impact of different resigning limitations (10, 20, 30) was tested, and the results of energy usage were compared between the ERR method and the RR and First-Fit methods. The results showed that power usage decreases as the resignation limit decreases and are represented in a graphical form in Fig. 12, with First-Fit as the baseline.

TABLE III. POWER UTILIZATION ERR AND FIRST-FIT POWER SAVE RESULTS

	Estimated Power (W)	**Power Meters Provide Actual Power**
ERR	744.53	695.20
Save with First-Fit Power	698.30	659.28
Improvements were made.	46.23	35.92

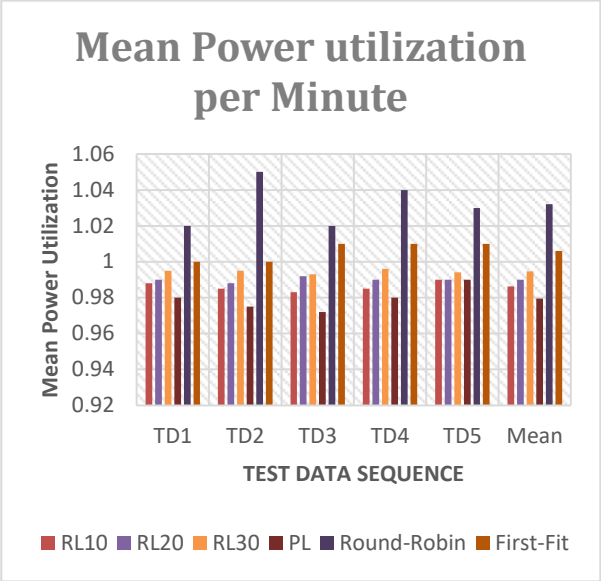

Fig. 12. Using ERR to calculate mean power utilization under different limits of resigning.

This simulation compares five different energy-saving solutions (Best-Fit, RR, First-Fit, ERR, and Fusion Method) by considering metrics like mean power use, mean PM count, and migration count. During busy hours, the Fusion Methodology is used. A graphical analysis is shown in Fig. 13 which demonstrates that the Fusion Methodology performs better in terms of energy conservation compared to the other approaches, as it uses PM loads as the resignation limit.

The paper compares the performance of five different algorithms for VM Scheduling in Eucalyptus cloud: RR, Greedy, PowerSave (similar to First-Fit), ERR and Fusion Method. The analysis compares the number of powered-on PMs and power consumption for each of these algorithms, with the results presented in Fig. 14 and 15. The paper shows that the use of the recommended strategies (ERR and Fusion Method) leads to a significant reduction in the number of powered-on PMs and power usage compared to the three basic Eucalyptus scheduling algorithms (RR, Greedy, and PowerSave).

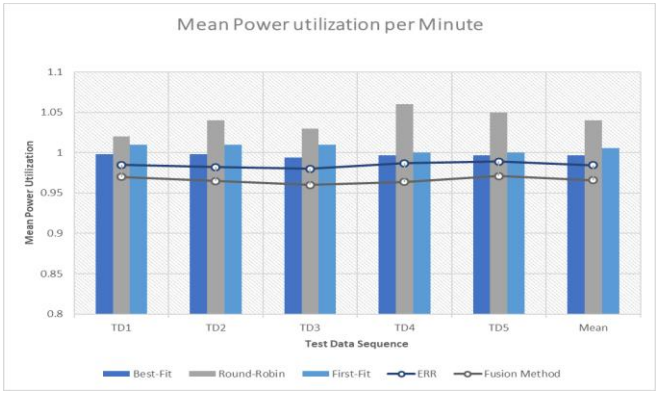

Fig. 13. Comparing the proposed technique's mean power utilization to that of other techniques.

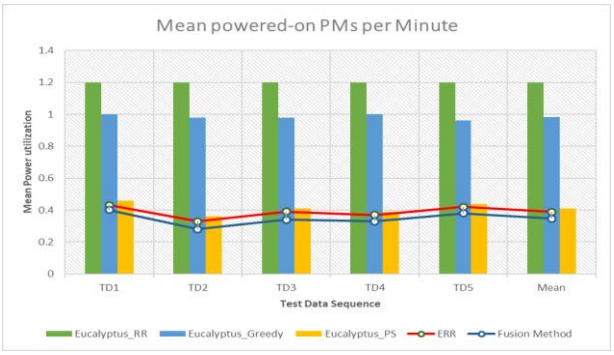

Fig. 14. Usage of RR base to analyze mean powered-on PMS.

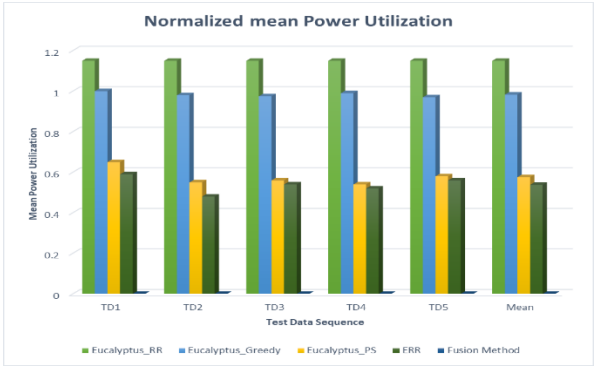

Fig. 15. The usage of RR base to analyse the normalised mean power utilisation.

A. *Eucalyptus with Walrus*

This section discusses the private cloud patterns controlled by Eucalyptus and the use of WALRUS, a data storage service for customer data. The Eucalyptus web interface supports "admin" and "user" accounts, and after registration, customers receive X509 certificates, secret key, and Query Id. The client credentials, including RSA private and public keys and X509 certificates, are stored in a file called "eucarc". WALRUS can be accessed through SOAP or REST via HTTP with the help of various utilities and is managed by ACLs and client credentials [10] [29] as shown in Fig. 16.

WALRUS is a data storage service used in Eucalyptus and can be accessed using tools like s3cmd, s3curl, s3fs, cloud berry s3 via SOAP or REST via HTTP. Access to data stored in WALRUS is managed by Access Control Lists and client credentials and secured using the MD5 hashing method. This section also mentions potential security attacks on private cloud systems powered by Eucalyptus that target cloud databases.

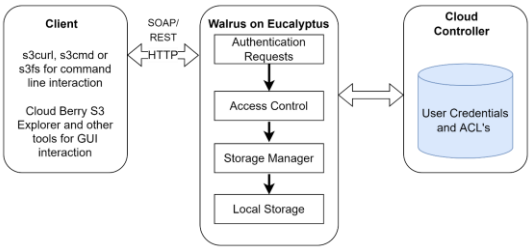

Fig. 16. Architectural demonstration of Eucalyptus-WALRUS.

1) Attacks related to buckets: The Eucalyptus private cloud system can be vulnerable to security attacks targeting cloud databases. Fig. 17 shows the credentials used for querying user interface names, such as EC2_SECRET_KEY and EC2_ACCESS_KEY, which are stored in the AUTH USERS table with attributes AUTH_USER_SECRETKEY, AUTH_USER_QUERY_ID, and AUTH_USER_NAME, respectively. One specific attack is the use of the file "eucarc," which is obtained from the eucalyptus auth.script catalogue. As shown in Fig. 18, an attacker must upload a new file called "eucarc" along with an S3 URL set that includes the IP address of the cloud controller and the values of EC2_SECRET_KEY and EC2_ACCESS_KEY, which are represented by AUTH_USER_SECRETKEY and AUTH_USER_QUERY_ID, respectively.

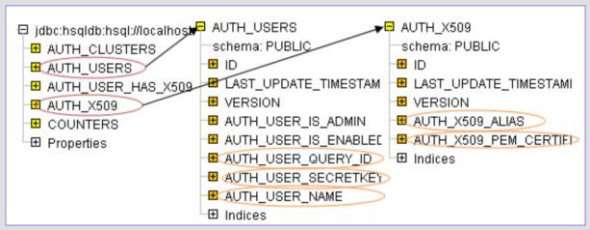

Fig. 17. Illustrations from the table script for Eucalyptus_auth.

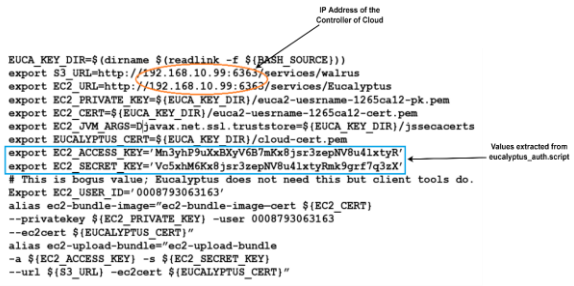

Fig. 18. The “eucarc”: credentials compressed file constituent.

The remaining parts of the "eucarc" paper might be omitted because they are not necessary for attacks involving buckets. The attacker essentially needs to obtain the most recent version of the eucarc document after it has been prepared and use the command s3curl to create a bucket that impersonates the client whose login details are used, or to gain access to a significant number of buckets that the client has reserved.

2) Attacks related to objects: Before launching attacks on an object, an intruder must be aware of the precise name of the bucket containing the target object. The eucalyptus-walrus can be used in two distinct ways to determine the precise name of a bucket. One possibility is the script catalogue. Fig. 19 shows how the names of the parent bucket, the individual objects, and the bucket owner are all shown in the OBJECTS table of this catalogue under the attributes OBJECT KEY, BUCKET NAME, and OWNER ID.

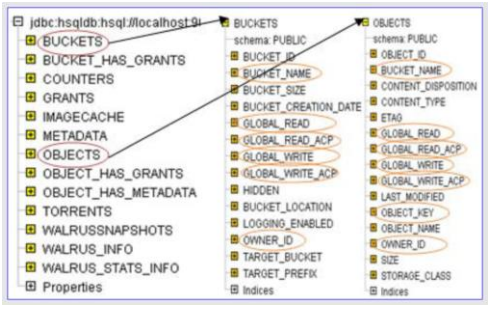

Fig. 19. Eucalyptus-walrus.script table illustrations.

The attacker must use the victim's interface certificates to create a new eucarc document and use s3curl to insert an object into the victim's bucket. The attacker must determine the object's size, MD5 checksum, and last update time and decide whether to read or delete the object. However, these actions require the owner to be an "administrator."

3) Attacks related to ACLs: In Eucalyptus, every WALRUS object and bucket has an associated Access Control List (ACL) in the form of a sub-resource. To launch ACL attacks, the attacker must first have access to ACL-related subresources. The ACL can be obtained using the s3curl command, and can be modified or a new one can be created with desired access control privileges. Attackers can also exploit the distribution of access control privileges to all cloud-registered users. The attacker can change the ACL file to grant access to all users by setting attributes in the eucalyptus walrus.script catalogue.

4) Attacks related to log file: Customers in Eucalyptus can create access logs for their buckets and decide where to send the logs. The logs can be treated like other objects, with the ability to list, remove, and read them. The logging data is stored in the eucalyptus walrus.script catalogue's properties TARGET PREFIX, TARGET BUCKET, and LOGGING ENABLED. If LOGGING ENABLED is set to TRUE, the TARGET PREFIX will add the client-specified prefix to the end of the log file names, and the TARGET BUCKET will be the client-selected bucket where the access log files will be saved. An attacker who gains access to the bucket containing the log entries can use the log entries as they see fit.

B. Analysis of Self-Adaptable System for Intrusion Detection

A data center with five physical nodes (1 controller, 1 network node, 2 compute nodes, and 1 LID host node) running Ubuntu Server 14.04 connected by a 1Gb/s network was set up on the Grid5000 platform for testing. Experiments were conducted using a memory-intensive workload with a 1024MB working set and 10 executions of the LMBench benchmark suite. The proposed framework's overhead during VM migration was tested with two new rule types related to ssh traffic.

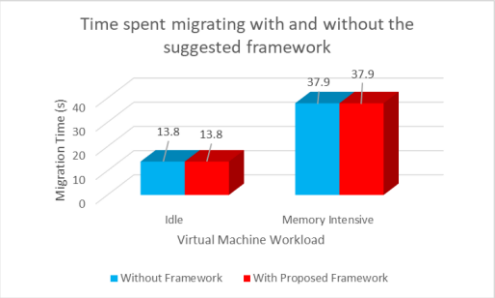

Fig. 20. Time spent migrating with and without the suggested framework.

The outcomes are displayed in Fig. 20. Our initial expectation that the proposed framework imposes minimal overhead on ordinary cloud operations is confirmed by the fact that the imposed overhead in both scenarios of an idle virtual machine and 0.0s represents a virtual machine with a memory-intensive workload. Fig. 21 and Fig. 22 display a breakdown of the two separate adaptation instances (new LIDS with traffic distribution and ruleset reconfiguration only) per phase.

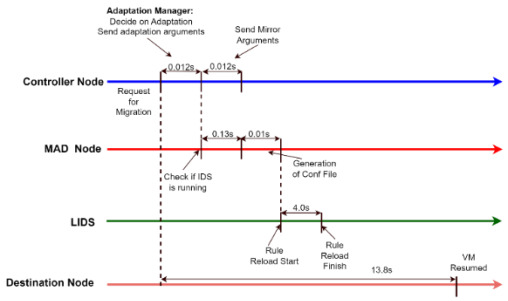

Fig. 21. Breakdown of the adaptation time when the proposed framework only modifies the imposed inside the LIDS ruleset.

Both instances involve the safety mechanism being in operation, however, when the plug vifs is called, the LIDS reconfiguration is finished significantly more quickly (4.14s and 0.97s respectively while the plug vifs function is called always after the 10th second).

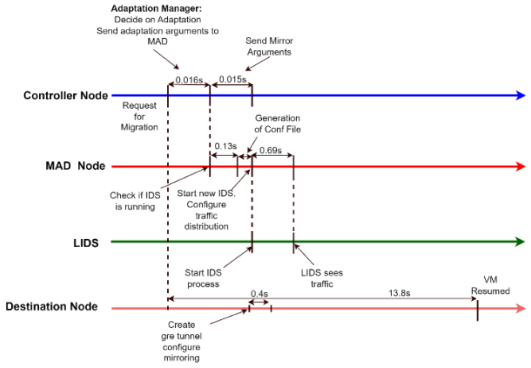

Fig. 22. A breakdown of how long it takes framework to build a mirroring tunnel, distribute traffic, and start a new LIDS.

The proposed framework reduces waiting time when restarting a virtual machine (VM) and completes a full adaptation cycle faster than migration. It takes 4.14 seconds to reconfigure the imposed ruleset in the first scenario and 0.97 seconds to access traffic in the second scenario when a new LID needs to be created. The total time needed by the framework is less than migration and creating a new IDS is easier than changing an old one.

Multiple LIDSs and MADs: A specialized script is created to replicate migration events by creating precise inputs from the Infrastructure Monitoring Probe for the Adaptation Manager. This allows for concurrent production of multiple adaptation events. During experimentation, a single instance of the Master Adaptation Driver (MAD) is configured to handle multiple LID instances, with the maximum number theoretically handled by a single MAD instance shown in Fig. 23.

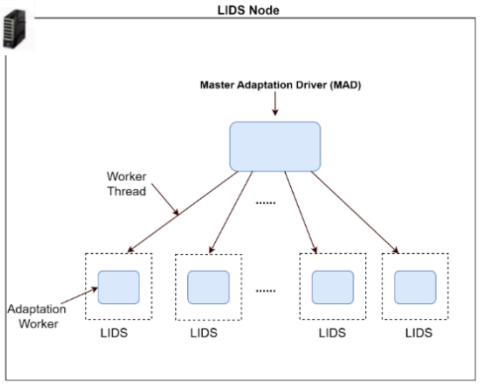

Fig. 23. Setup for MAD scalability.

Our findings demonstrate that up to 50 LIDS can be supported by a single MAD instance running on a dedicated node with 24GB RAM. Fig. 24 displays the MAD agent's typical response time under various LIDS loads.

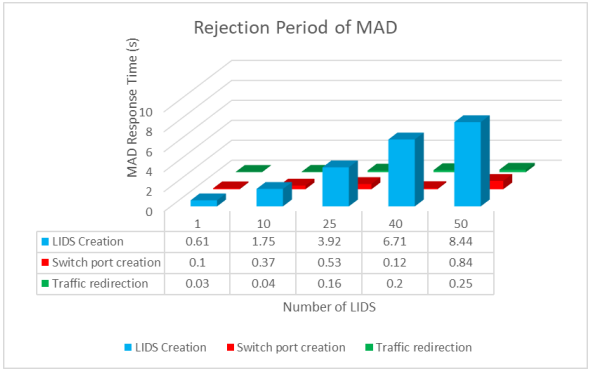

Fig. 24. MAD reaction periods.

The results show that the longest creation time for a new LID container requires interaction with the Docker daemon. The proposed framework can handle 50 simultaneous LID launching requests and still has a faster reaction time than the average migration time for an idle VM. The framework does not include the time for building the new LID configuration file or testing its functionality due to their low impact on the overall duration. Each LID is typically assigned one core in a production setting to maintain performance. The framework was tested with 10 concurrent adaptation requests to simulate a commercial setting. The maximum number of LID's that a single Master Adaptation Driver instance can manage

simultaneously was determined, and there is room for nearly 100 worker threads for the Adaptation Manager. One AM instance is configured to control multiple MADs in Fig. 25.

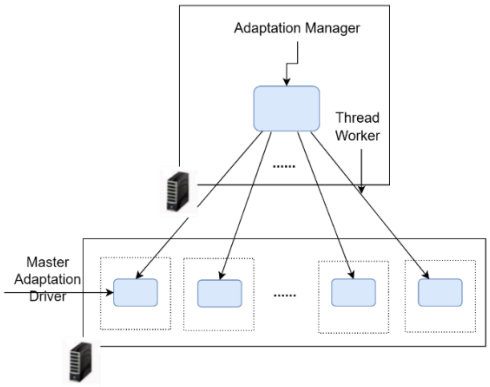

Fig. 25. AM scaling configuration.

The chosen monitoring mechanism for a single VM is a single LID. The framework was tested by simulating 50 dynamic events for 50 different VMs, providing adaptation requirements for 50 LID of each thread. A worker thread relocates all of its virtual machines to the same destination node to target the LID under the same MAD instance it is handling. The worker thread parses the vm info.xml file containing all the VM-related data to obtain settings for each of the 50 VMs it is responsible for. The minimum number of VM entries in the vm info.xml file required is calculated as maximum AM worker threads multiplied by the number of VMs per thread (in this case 100 AM worker threads and 50 VMs per thread, requiring 5000 entries). Justifications for each LID adaptation are recorded in a separate file. The worker thread delivers 50 files, one for each LID, to the MAD in charge of those 50 LID after establishing a secure connection. Only one file is required in the experiment as all virtual machines for a single worker thread are moved to the same compute node. The experiment aims to understand how the AM scales with the quantity of MADs, not the number of compute nodes.

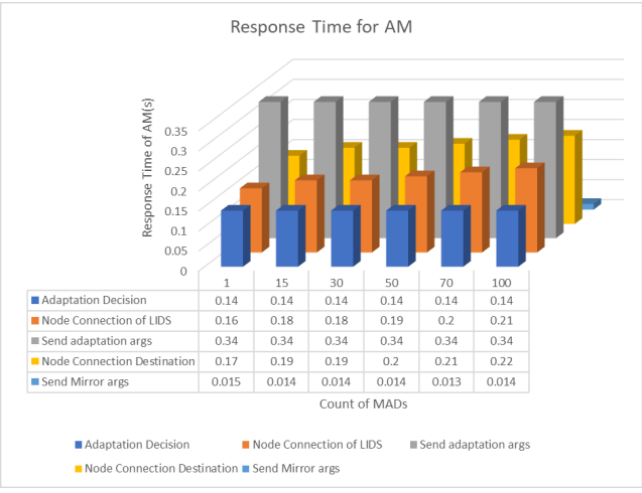

Fig. 26. AM response time.

Fig. 26 presents the outcomes. As the data show, the formation of the secure connection is the phase that is most impacted by raising the MADs' load for the AM. This is due to the fact that each MAD has its own IP address and is kept in a distinct container, necessitating the use of a separate secure connection. On the AM side, we track the time it takes to send the adaption arguments. No network contention-related delay is seen in the outcome since we don't wait for each instance to validate that it has received the files. However, because each MAD instance is essentially run on a separate container on the same node, there may be many processes running on the node that are causing severe ssh connection formation delays. Since each MAD instance would operate in a distinct, less-loaded node in a real-world scenario, the findings of our experiment are unsatisfactory. The adaption decision time is not greatly shortened by the multi-threading method because all VM-related data is retained in a single file.

Up to 5000 LIDS instances can be managed by a single AM instance and still respond to thread requests in under one second, according to the results. The testbed's memory capacity is the single factor limiting the number of LIDS instances that can be employed for our research. If framework is implemented in a different configuration with production nodes having memory capacities that are substantially a minimum of 24 GB of RAM per node, the number of instances could rise.

The pidstat programme from the sysstat suite, a utility used to measure the resource utilisation of a given job running in an OS, is utilized to compute the resource consumption of an AM in terms of CPU and memory handling multiple MADs. Each experiment asks the first worker thread to run pidstat as soon as the adaption parameters are received, and the monitoring is stopped after the last worker thread has finished its task. This method ensures that during the adaptation process, we will only calculate the resource usage of each worker thread. No other framework-related processes use resources since the worker thread in charge of that adaptation request handles all the adaptation-related duties in that request. One second was chosen as the monitoring interval. The results are shown in Table IV. The Graphical illustration of resources consumption by AM is analysed in the Fig. 27.

The CPU use grows as the number of AM worker threads rise since there is a one-time CPU cost for starting a new ssh session. For each worker thread, the measurements compute the worst-case situation, which is to create a new connection. When the framework has to modify an existing LIDS, it can transfer the file containing the adaption arguments using an already established connection as a result of the anticipated decrease in CPU usage.

TABLE IV. CONSUMPTION OF RESOURCES BY THE AM COMPONENT

Number of MADs	Usr%	Sys%	CPU%	Memory (MB)
15	18.15	2.35	20.35	189.66
30	24.30	3.39	27.08	189.83
50	25.19	3.71	30.12	189.74
70	27.63	3.82	31.98	189.61
100	29.4	3.98	33.45	189.89

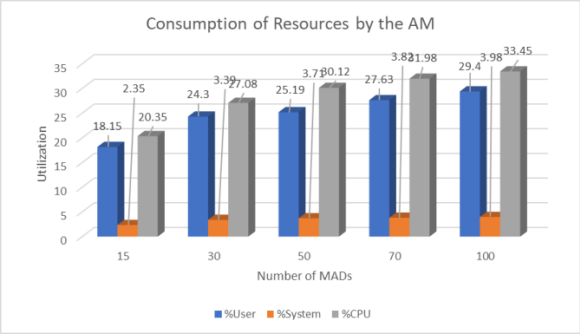

Fig. 27. Graphical illustration of resources consumption by AM.

The proposed framework for self-adapting security monitoring in cloud infrastructure is better than previous approaches for several reasons as below

1) Customization for each tenant: The framework takes into account the specific security monitoring requirements of each tenant, allowing for customized security monitoring configurations. This is an improvement over previous approaches that had a one-size-fits-all security monitoring approach that may not have been optimal for all tenants.

2) Self-adaptation: The framework includes an Adaptation Manager that can adapt to changing conditions in the cloud infrastructure and serve as a coordinator of the adaptation process. This allows the framework to respond dynamically to events in the cloud environment and adjust security monitoring configurations as needed. This is an improvement over previous approaches that did not have a self-adapting mechanism and required manual adjustments.

3) Cost efficiency: The framework aims to achieve cost efficiency by sharing monitoring resources between the tenants and the provider. The equitable sharing of monitoring resources is established with an adjustable threshold mentioned in the SLA. This is an improvement over previous approaches that may have been resource-intensive and costly.

4) Comprehensive solution: The framework provides a comprehensive solution that combines precise security monitoring with self-adaptation. The design of the framework ensures that it does not introduce new security weaknesses or affect the performance of the infrastructure. This is an improvement over previous approaches that may have been limited in scope or may have introduced new security weaknesses.

Overall, the proposed framework is an improvement over previous approaches because it is customized for each tenant, includes a self-adapting mechanism, achieves cost efficiency, and provides a comprehensive solution for security monitoring in cloud infrastructure.

V. Conclusion and Future Works

In this paper, the increasing number of tasks in clouds due to virtualization and cloud computing technology adoption is discussed. VM scheduling strategies are important for determining the allocation of cloud resources to handle these tasks for maintaining acceptable throughput and revenue. The paper evaluates the current knowledge on legacy methods and specific virtual machine scheduling algorithms for the Eucalyptus cloud environment and compares some existing algorithms using specific measures for a better understanding.

The Eucalyptus cloud uses two methods for scheduling VMs: Fusion Method and Enhanced Round-Robin. The experiment showed that Enhanced Round-Robin uses less energy compared to other methods and that using Physical Machine load as a limit of resigning saves the most energy. The Fusion Method outperforms other techniques by reducing the number of Physical Machines turned on and increasing power efficiency. The authors also developed a self-adapting security monitoring system with goals of security, cost savings, tenant-driven customization, and self-adaptation. The Adaptation Manager is the main element that can adapt to changing conditions in the cloud infrastructure and serves as a coordinator of the adaptation process.

This paper presents a self-adapting security monitoring system for cloud infrastructure that takes into account the specific monitoring requirements of each tenant. The system uses Master Adaptation Drivers to convert the tenant requirements into configuration settings and the Adaptation Manager to coordinate the adaptation process. The framework ensures security, cost efficiency, and responsiveness to dynamic events in the cloud environment. The design of the framework is such that it does not introduce new security weaknesses or affect the performance of the infrastructure. The system provides a comprehensive solution that combines precise security monitoring with self-adaptation.

The current security monitoring platform needs improvement to support more types of monitoring devices such as network traffic analysis and inside-the-host activity monitoring. The platform is also limited to supporting firewall functionality and doesn't cover the consequences of multi-tenant setups. The future work includes incorporating log collectors and aggregators and addressing the needs of a super-tenant (the provider) in the security monitoring architecture. The equitable sharing of monitoring resources between the tenants and the provider should be established with an adjustable threshold mentioned in the SLA.

Acknowledgment

The authors would like to thank Malaysia University of Science and Technology, for the support and encouragement to carry out this study.

References

[1] P. C. Yang, J. H. Chiang, J. C. Liu, Y. L. Wen, and K. Y. Chuang, “An efficient cloud for wellness self-management devices and services,” Proc. - 4th Int. Conf. Genet. Evol. Comput. ICGEC 2010, pp. 767–770, 2010, doi: 10.1109/ICGEC.2010.194.

[2] S. Zhang, S. Zhang, X. Chen, and X. Huo, “The comparison between cloud computing and grid computing,” ICCASM 2010 - 2010 Int. Conf. Comput. Appl. Syst. Model. Proc., vol. 11, 2010, doi: 10.1109/ICCASM.2010.5623257.

[3] N. Sadashiv and S. M. D. Kumar, “Cluster, grid and cloud computing: A detailed comparison,” ICCSE 2011 - 6th Int. Conf. Comput. Sci. Educ. Final Progr. Proc., pp. 477–482, 2011, doi: 10.1109/ICCSE.2011.6028683.

[4] E. Raggi, K. Thomas, T. Parsons, A. Channelle, and S. van Vugt, "Social Networks and Cloud Computing," Begin. Ubuntu Linux, pp. 337–348, 2010, doi: 10.1007/978-1-4302-3040-3_15.

[5] "The Treacherous 12 Top Threats Working Group," 2016. [Online]. Available: https://cloudsecurityalliance.org/download/the-treacherous-twelve.

[6] Dave Shackleford, "9 cloud migration security considerations and challenges | TechTarget," Voodoo Security, Nov. 23, 2021. https://www.techtarget.com/searchcloudcomputing/tip/9-cloud-migration-security-considerations-and-challenges (accessed Feb. 01, 2023).

[7] S. A. Bello et al., "Cloud computing in construction industry: Use cases, benefits and challenges," Autom. Constr., vol. 122, p. 103441, Feb. 2021, doi: 10.1016/J.AUTCON.2020.103441.

[8] I. M. Khalil, A. Khreishah, and M. Azeem, "Cloud Computing Security: A Survey," Comput. 2014, Vol. 3, Pages 1-35, vol. 3, no. 1, pp. 1–35, Feb. 2014, doi: 10.3390/COMPUTERS3010001.

[9] M. Carroll, A. Van Der Merwe, and P. Kotzé, "Secure cloud computing: Benefits, risks and controls," 2011 Inf. Secur. South Africa - Proc. ISSA 2011 Conf., 2011, doi: 10.1109/ISSA.2011.6027519.

[10] A. Waqar, A. Raza, and H. Abbas, "User Privacy Issues in Eucalyptus: A Private Cloud Computing Environment," in 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, Nov. 2011, pp. 927–932. doi: 10.1109/TrustCom.2011.128.

[11] M. F. Manzoor, A. Abid, M. S. Farooq, N. A. Nawaz, and U. Farooq, "Resource Allocation Techniques in Cloud Computing: A Review and Future Directions," Elektron. ir Elektrotechnika, vol. 26, no. 6, pp. 40–51, Dec. 2020, doi: 10.5755/J01.EIE.26.6.25865.

[12] L. Wang and D. Zhang, "Research on OpenStack of open source cloud computing in colleges and universities' computer room," IOP Conf. Ser. Earth Environ. Sci., vol. 69, no. 1, p. 012140, Jun. 2017, doi: 10.1088/1755-1315/69/1/012140.

[13] M. E. Suliman, "A Brief Analysis of Cloud Computing Infrastructure as a Service (IaaS)," 2021. [Online]. Available: www.ijisrt.com.

[14] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proc. IEEE, vol. 103, no. 1, pp. 14–76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[15] McKeownNick et al., "OpenFlow," ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008, doi: 10.1145/1355734.1355746.

[16] H. Wang, A. Srivastava, L. Xu, S. Hong, and G. Gu, "Bring your own controller: Enabling tenant-defined SDN apps in IaaS clouds," Proc. - IEEE INFOCOM, Oct. 2017, doi: 10.1109/INFOCOM.2017.8057137.

[17] M. Johns, "Code-injection vulnerabilities in web applications - Exemplified at cross-site scripting," IT - Inf. Technol., vol. 53, no. 5, pp. 256–260, Sep. 2011, doi: 10.1524/ITIT.2011.0651/MACHINEREADABLECITATION/RIS.

[18] T. Grandison and E. Terzi, "Intrusion Detection Technology," Encycl. Database Syst., pp. 1568–1570, 2009, doi: 10.1007/978-0-387-39940-9_209.

[19] S. Lata and D. Singh, "Intrusion detection system in cloud environment: Literature survey & future research directions," Int. J. Inf. Manag. Data Insights, vol. 2, no. 2, p. 100134, Nov. 2022, doi: 10.1016/j.jjimei.2022.100134.

[20] D. Nurmi et al., "The Eucalyptus Open-source Cloud-computing System."

[21] H. Alshaer, "An overview of network virtualization and cloud network as a service," Int. J. Netw. Manag., vol. 25, no. 1, pp. 1–30, Jan. 2015, doi: 10.1002/NEM.1882.

[22] M. Yassin, H. Ould-Slimane, C. Talhi, and H. Boucheneb, "Multi-Tenant Intrusion Detection Framework as a Service for SaaS," IEEE Trans. Serv. Comput., vol. 15, no. 05, pp. 2925–2938, Sep. 2022, doi: 10.1109/TSC.2021.3077852.

[23] A. Giannakou et al., "Self-adaptable Security Monitoring for IaaS Cloud Environments," Jul. 2017, Accessed: Feb. 05, 2023. [Online]. Available: https://hal.inria.fr/tel-01653831.

[24] I. C. Lin, C. C. Chang, and C. H. Peng, "An Anomaly-Based IDS Framework Using Centroid-Based Classification," Symmetry 2022, Vol. 14, Page 105, vol. 14, no. 1, p. 105, Jan. 2022, doi: 10.3390/SYM14010105.

[25] G. Chen et al., "Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services."

[26] C. C. Lin, P. Liu, and J. J. Wu, "Energy-efficient virtual machine provision algorithms for cloud systems," Proc. - 2011 4th IEEE Int. Conf. Util. Cloud Comput. UCC 2011, pp. 81–88, 2011, doi: 10.1109/UCC.2011.21.

[27] S. Knox, P. Meier, J. Yoon, and J. J. Harou, "A python framework for multi-agent simulation of networked resource systems," Environ. Model. Softw., vol. 103, pp. 16–28, May 2018, doi: 10.1016/J.ENVSOFT.2018.01.019.

[28] T. Madi et al., "ISOTOP," ACM Trans. Priv. Secur., vol. 22, no. 1, Oct. 2018, doi: 10.1145/3267339.

[29] A. Rath, B. Spasic, N. Boucart, and P. Thiran, "Security Pattern for Cloud SaaS: From System and Data Security to Privacy Case Study in AWS and Azure," Comput. 2019, Vol. 8, Page 34, vol. 8, no. 2, p. 34, May 2019, doi: 10.3390/COMPUTERS8020034.